View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by CERN Document Server

PROCEEDINGS OF SCIENCE

Application of Gene Expression Programming to Event Selection in High Energy Physics

Liliana Teodorescu*
Brunel University
E-mail: Liliana.Teodorescu@brunel.ac.uk

Ivan D. Reid
Brunel University
E-mail: Ivan.Reid@brunel.ac.uk

Gene Expression Programming is a new evolutionary algorithm found to be very efficient for solving benchmark problems from computer science. The algorithm was successfully tested for event selection in high energy physics for the $K_S \to \pi^+\pi^-$ decay process. This paper presents an extended version of this analysis, as well as a comparison of its results with those obtained with an Artificial Neural Network and a Boosted Decision Trees method. All three methods produced selection functions which allowed good signal and background separation for the problem investigated, with the classification accuracies higher than 95%.

XI International Workshop on Advanced Computing and Analysis Techniques in Physics Research
April 23-27 2007
Amsterdam, the Netherlands

*Speaker.

© Copyright owned by the author(s) under the terms of the Creative Commons Attribution-NonCommercial-ShareAlike Licence.

PoS(ACAT)051

1. Introduction

In order to address the data analysis challenges imposed by the complexity of the data generated by current and future high energy physics experiments, new techniques for performing various analysis tasks need to be investigated.

A class of algorithms less explored in High Energy Physics is the Evolutionary Algorithms class. These are computer algorithms inspired by the theories of the natural evolution of the species. Natural evolution, in this context, is defined as the optimisation process that aims to increase the ability of individuals to survive and reproduce in a specific environment. This ability is quantitatively measured by the evolutionary fitness of the individuals. The unique characteristics of each individual are represented in its chromosomes. Through the natural selection process the fitter chromosomes will mate more often, creating offspring with similar or better fitness. The goal of the natural evolution process is to create a population of increasing fitness.

Evolutionary Algorithms use simulation of natural evolution on a computer. The candidate solution of the problem to be solved by the algorithm represents an individual that is encoded in a form understood by the computer called a chromosome. A chromosome can be divided into one or more constituent parts called genes. The quality of the candidate solution is evaluated with a function called fitness function. The reproduction process of the individuals is simulated by applying on them a set of operators, called genetic operators, creating genetic variation. The selection of the individuals for reproduction is, usually, proportional to their fitness (value of the fitness function for the individual). Through an iterative process, the algorithm improves the quality of the solution until an optimal solution is found.

The best known versions of Evolutionary Algorithms are Genetic Algorithms (GA) [1] and Genetic Programming (GP) [2]. The main difference between the two algorithms is the encoding method of the candidate solution into a chromosome. In GA the chromosome is, in the classical version of the algorithm, a binary string of fixed length while in GP it is a LISP expression of fixed length represented graphically as a tree. Genetic variation is produced by applying genetic operators on the binary string or on the tree. For GP this method generates many syntactically incorrect structures during the reproduction process, resulting in high inefficiencies of the algorithms.

A new variant of the Evolutionary Algorithms, called Gene Expression Programming (GEP) [3] was proposed in 2001. It overcomes some of the limitations of GA and GP by working with two entities, the chromosome and the expression tree (ET). The ET corresponds to a mathematical expression that represents the actual candidate solution. The chromosome is the encoder of the candidate solution which is then translated into an expression tree.

The first application of GEP to particle physics data analysis was for an event selection problem for the $K_S \to \pi^+\pi^-$ decay process and was presented in [5] and [6]. This paper extends that analysis, investigating the influence of the number of the input variables and the input functions, and of the number of events in the data sample on the algorithm results. Also, the effect of a parsimony pressure applied to the fitness function was studied. The results obtained with GEP were compared with those obtained with an Artificial Neural Network (ANN) and a Boosted Decision Trees (BDT) algorithm.

PoS(ACAT)051

2. Event Selection Problem

The three algorithms, GEP, ANN and BDT, were tested for an event selection problem. Using a statistical learning approach, they were used to extract selection criteria for a signal/background classification. The purpose of the investigation was the study of the behaviour and performance of the algorithms, rather than to extract a physics result.

The physics process used for this study was the decay process $K_S \to \pi^+\pi^-$ with K_S produced in e^+e^- interaction at $\approx$10 GeV in the BaBar experiment [7]. The data samples were Monte Carlo events. $e^+e^- \to q\bar{q}$ events (q being a quark and $\bar{q}$ an antiquark) were generated using JETSET [8] (for q being the u, d, s and c quarks) and EvtGen [9] (for q being the b quark) simulation packages. The generated events were passed through the detector response simulation package [10] and reconstructed with the BaBar analysis software. The reconstructed K_S candidates were considered signal if they were associated with the generated K_S particles and their reconstructed π daughters were associated with the π daughters of the generated K_S particles. All the other reconstructed K_S candidates were considered background.

Training and test samples of equal sizes (5,000 or 10,000 events) with varying signal-to-background (S/B) ratios (1:1, 1:2, 1:4 and 1:10) and with either 8 or 20 event variables were used as input to the analysis. Selection models were extracted from the training data samples and tested on the corresponding test data samples.

The set of the 8 event variables contained the variables usually used in a standard cut-based analysis for the K_S decay process:

- *doca* - distance between the two π daughters of K_S at the point of closest approach,
- R_{XY}- radius of the cylinder that defines the e^+e^- interaction region,
- $|R_Z|$ - half length of the cylinder that defines the e^+e^- interaction region,
- $|\cos(\theta_{\rm hel})|$ - absolute value of the cosine of the K_S helicity angle,
- *SFL* - K_S signed flight length defined as the projection of the vector from interaction point to K_S decay vertex on the K_S momentum direction,
- *Fsig* - statistical significance of the K_S flight length,
- *Pchi* - χ^2 probability of K_S vertex,
- *Mass* - K_S reconstructed mass.

The set of 20 variables contained, in addition of those listed before, vertex variables of the K_S particle and kinematic variables of the $\pi^\pm$ particles:

- V_{XX},V_{XY},V_{XZ} - cartesian coordinates of K_S vertex,
- $\cos(\theta_{K_S})$, ϕ_{K_S}, P_{K_S} - polar coordinates of the K_S momentum vector,
- $\cos(\theta_{\pi_1})$, ϕ_{π_1}, P_{π_1} - polar coordinates of the π^+ momentum vector,
- $\cos(\theta_{\pi_2})$, ϕ_{π_2}, P_{π_2} - polar coordinates of the π^- momentum vector.

PoS(ACAT)051

3. Gene Expression Programming method

3.1 Algorithm

The GEP algorithm is described in detail in [4] and an extensive summary is also presented in [5].

The algorithm starts with the problem definition, the encoding of the candidate solution of the problem into a chromosome and the definition of the fitness function that describes how good the candidate solution is for the problem at hand. Then an initial population of chromosomes is randomly generated, the chromosomes are translated into expression trees and then into a mathematical expression, and the fitness function is evaluated for each chromosome. If a solution of adequate quality is not found, a set of chromosomes is selected and reproduced, creating a new generation of chromosomes. The process is repeated until an optimal solution to the problem is found or a given number of generations have passed.

The candidate solution is encoded into a chromosome composed of one or more genes of equal length. A gene is divided into a head composed of terminals (variables and constants) and functions, and a tail composed only of terminals. The length of the head (h) is an input parameter of the algorithm while the length of the tail (t) is given by:

$$t = h(n-1)+1 \tag{3.1}$$

where n is the largest arity of the functions used in the gene head.

The list of functions and variables to be used in a gene is input information for the algorithm, while the constants are created by the algorithm itself in a range specified by the user.

Each gene is translated into an expression tree using simple rules as described in [4]. In the case of multigenic chromosomes, the expression trees are connected with a linking function defined by the user. The selection of the chromosomes to be reproduced is made using the elitism (unchanged replication of the best fitted chromosome into the next generation) and the roulette-wheel [11] methods. The chromosomes selected with the roulette-wheel method are reproduced by applying on them a set of genetic operators:

- mutation - randomly changes an element of a chromosome into another element (preserving the rule that the tails contain only terminals);
- transposition - randomly copies a part of the chromosome to another point in the gene head of the same chromosome;
- recombination (cross-over) - exchanges parts of a pair of randomly chosen chromosomes.

In this study three sets of functions, containing 18 comparison functions (listed in Table 1a), 38 mathematical functions (listed in Table 1b) and 56 functions (the combination of the previous two sets) were used as input to the algorithm for chromosome development. The constants used in the chromosome are developed by the algorithm in the range of $(-10,10)$ (given as input to the algorithm).

Other GEP input parameters were: the length of the gene head (between 1 and 20), the number of chromosomes per generation (100) and the maximum number of generations (between 3,000 and

PoS(ACAT)051

Table 1a 18 Comparison functions		Table 1b 38 Mathematical functions			
Function	Definition	Funct.	Def.	Funct.	Def.
OR1	if x<0 OR y<0, 1, else 0	+	(x+y)	-	(x-y)
OR2	if x≥0 OR y≥0, 1, else 0	*	(x*y)	/	(x/y)
OR3	if x≤0 OR y≤0, 1, else 0	Mod	mod(x,y)	Pow	pow(x,y)
OR4	if x<1 OR y<1, 1, else 0	Sqrt	sqrt(x)	Exp	exp(x)
OR5	if x≥1 OR y≥1, 1, else 0	Pow10	pow(10,x)	Ln	ln(x)
OR6	if x≤1 OR y≤1, 1, else 0	Log	log(x)	Abs	abs(x)
AND1	if x<0 AND y<0, 1, else 0	Inv	1/x	Neg	-x
AND2	if x≥0 AND y≥0, 1, else 0	Sin	sin(x)	Cos	cos(x)
AND3	if x≤0 AND y≤0, 1, else 0	Tan	tan(x)	Csc	csc(x)
AND4	if x<1 AND y<1, 1, else 0	Sec	sec(x)	Cot	cot(x)
AND5	if x≥1 AND y≥1, 1, else 0	Asin	arcsin(x)	Acos	arccos(x)
AND6	if x≤1 AND y≤1, 1, else 0	Atan	arctan(x)	Acsc	arccsc(x)
LT2B	if x<y, 1, else 0	Asec	arxsec(x)	Acot	arccot(x)
GT2B	if x>y, 1, else 0	Sinh	sinh(x)	Cosh	cosh(x)
LOE2B	if x≤y, 1, else 0	Tanh	tanh(x)	Csch	csch(x)
GOE2B	if x≥y, 1, else 0	Sech	sech(x)	Coth	coth(x)
ET2B	if x=y, 1, else 0	Asinh	arcsinh(x)	Acosh	arccosh(x)
NET2B	if x≠y, 1, else 0	Atanh	arctanh(x)	Acsch	arccsch(x)
		Asech	arxsech(x)	Acoth	arccoth(x)

Table 1: The sets of GEP input functions

20,000, depending on the complexity of the chromosomes). The genetic operator rates were kept as recommended in [4]: 0.044 for mutation, 0.3 for transposition and 0.1 for recombination.

The fitness function was the number of hits (the number of events correctly classified as signal or background).

The performance of the algorithm was analysed in terms of the classification accuracy parameter (*Acc*) defined as the the ratio of the total number of events correctly classified (signal and background) to the total number of events of the sample.

The GEP implementation in the Windows package GeneXProTools 4.0 [12] was used. Typical performance was 1-4 generations/second, depending on the complexity of the model (3.2 GHz Pentium 4 Mobile).

PoS(ACAT)051

3.2 Analysis and Results

3.2.1 Model Evolution

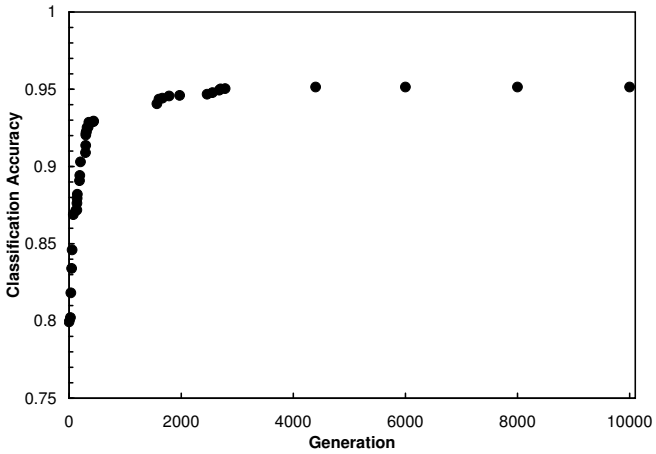

Figure 1: Evolution of GEP training accuracy over 10,000 generations for a 5,000-event data sample, S/B=1:4, 8 variables, head size 10 and 38 input functions.

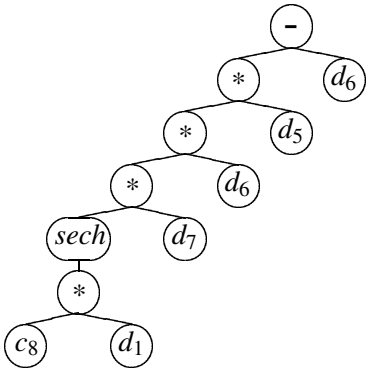

Figure 2: The expression tree for the best model developed during the evolution depicted in Fig. 1 , where d_0-d_7 are the event variables and c_8 is a constant. Each function, variable and costant is represented as a node in the tree.

The GEP models were monitored as the population evolved in order to identify the convergence of the search process. An example is presented in Fig. 1 that shows how the training accuracy improved as one population was followed for 10,000 generations. In this particular run, the best solution was found after 4,399 generations and remained unchanged until the end of the run. The expression tree for the best-adapted chromosome is given in Fig. 2.

3.2.2 Model Complexity

The dependence of the models developed by GEP as a function of the complexity of the chromosomes was studied for different input functions and input data sample configurations. The length of the gene head was varied between 1 and 30 and the classification accuracy obtained with

PoS(ACAT)051

the fittest chromosome was recorded. It was observed that a high quality solution is obtained with chromosomes with the gene head length equal to 10 and no significant improvements were obtained with more complex chromosomes. An example is shown in Fig. 3 that illustrates the change in the training accuracy and the testing accuracy as the size of the gene head is varied, for a data sample with 5,000 events, 8 input variables and signal-to-background ratio of 1:4. It is seen that a plateau in the classification accuracy has been reached by gene head length equal to 10.

For the rest of the study a gene head length of 10 was considered the optimal chromosome configuration for the problem investigated in this paper. All the results reported below were obtained with this configuration.

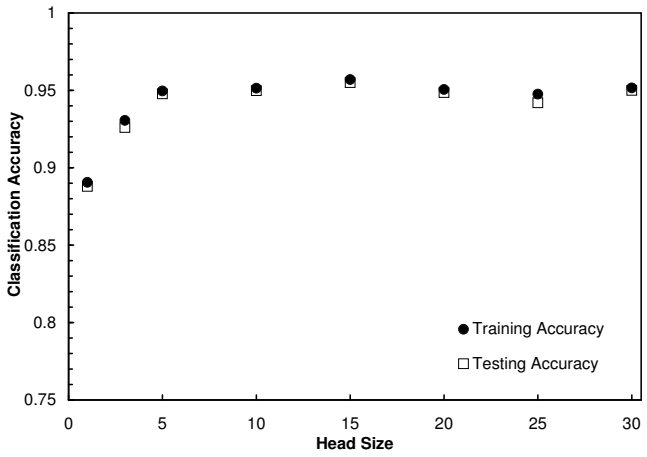

Figure 3: GEP training accuracy and associated testing accuracy for a 5,000-event data sample, S/B=1:4, 8 variables, 38 input functions, as a function of head size.

3.2.3 Input functions and parsimony pressure

The influence of the number and type of input functions on the performance of the selection models developed by GEP was studied with a data sample containing 8 variables, 5,000 events and having signal-to-background ratio equal to 1:4.

The analysis was repeated, in the same conditions, with the set of 18 functions (with which cut-type criteria are built) listed in Table 1a, the set of 38 mathematical functions listed in Table 1b and the set of 56 functions obtained combining the two previous sets.

The results, in terms of the classification accuracy (and its statistical uncertainty) on the training and test data samples, and the dimension of the models (number of nodes in the expression tree) are listed in Table 2. The statistical uncertainty of the classification accuracy was calculated with the relation $\sqrt{A_{cc}(1-A_{cc})N}$, where A_{cc} is the classification accuracy and N is the number of events in the sample. The results presented in Table 2 indicate no significant variation of the classification accuracy with the number of input functions. Both the comparison functions and the mathematical functions, as well as the combination of the two, produce solutions of similar quality. As the comparison functions provide good solutions that can be also easily interpreted from physics point of view, they can be considered the optimal input functions for the problem studied here. Addition of new input functions does not extend the potential of the algorithm. This behaviour is common to all

PoS(ACAT)051

Fitness Method	Classification accuracy			Classification accuracy with parsimony pressure		
	Accuracy (%)		No. of	Accuracy (%)		No. of
Functions	Train	Test	Nodes	Train	Test	Nodes
18	95.1 ± 0.3	95.1 ± 0.3	17	94.8 ± 0.3	94.8 ± 0.3	13
38	95.1 ± 0.3	95.0 ± 0.3	12	95.4 ± 0.3	95.1 ± 0.3	14
56	95.3 ± 0.3	95.4 ± 0.3	11	95.5 ± 0.3	95.1 ± 0.3	14

Table 2: The effect of the number. of input functions and of parsimony pressure on classification accuracy and expression tree size (no. of nodes). (S/B=1:4, 5,000 events, 8 variables, head size=10)

evolutionary algorithms. They provide the best solution with a chromosome of a certain complexity or a certain amount of input information, beyond of which no improvement is achieved.

The analysis was repeated applying a parsimony pressure [4] to the fitness function, in an attempt to reduce the complexity of the models. The fitness-with-parsimony-pressure is given by the following expression:

$$fpp = rf(1 + \frac{1}{5000}\frac{S_{\max} - S}{S_{\max} - S_{\min}}) \tag{3.2}$$

where rf is the raw fitness (number of hits, in this study), S is the size of ET (number of nodes), and $S_{\max}$ and $S_{\min}$ are the maximum and minimum ET sizes,

$$S_{\max} = G(h+t) \tag{3.3}$$

$$S_{\min} = G \tag{3.4}$$

where G is the number of genes (in this case 1) and h and t are the chromosome head and tail sizes, respectively.

The results of this analysis are also presented in Table 2 which contains the classification accuracy on training and test data obtained with parsimony pressure, as well as the number of nodes of the expression trees. These results show no significant effect of applying parsimony pressure during GEP modelling. This pressure did not improve the accuracy of the models developed, nor indeed was it always successful in lowering the models' complexity.

3.2.4 Data complexity

The effect on GEP solution quality of changing the number of event variables from 8 to 20, and the size of the datasets from 5,000 to 10,000 was also studied for the comparison input functions. The results are summarised in Table 3. No significant change is seen in the classification accuracy when adding new event variables or new events in the data sample.

The similarity between the results obtained with 8 and 20 variables data sets indicates that GEP is able to ignore the irrelevant variables in the final solution. Irrelevant variables means, in this context, variables that do not contribute significantly to the signal/background separation. The

PoS(ACAT)051

Classification accuracy (%)							
5,000-event Samples				10,000-event Samples			
8 variables		20 variables		8 variables		20 variables	
Training	Test	Training	Test	Training	Test	Training	Test
95.1 ± 0.3	95.1 ± 0.3	94.9 ± 0.3	94.8 ± 0.3	94.9 ± 0.2	94.8 ± 0.2	95.1 ± 0.2	95.0 ± 0.2

Table 3: The effect on GEP classification accuracy of varying the number of event variables and the sample size for S/B=1:4 data, head size=10, 18 input functions

PoS(ACAT)051

initial set of 8 variables are, indeed, the most powerful, as it was seen in a standard cut-based analysis of the physics process studied here [13].

The similarity between the results obtained with 5,000 and 10,000 events indicates that, at least for the problem studied here, GEP is able to find high quality solutions with relatively small amounts of data.

4. Artificial Neural Network method

4.1 Algorithm

The Artificial Neural Network used in this study is a Multi-Layer Perceptron consisting of an input layer of N neurons (where N is the number of event variables), two hidden layers of N-1 and N-2 neurons, respectively, and an output layer of one neuron. The input to each hidden-layer or output neuron j is a weighted linear combination with bias of the outputs y_i from the previous layer's neurons:

$$x_j = w_{0j} + \sum_i w_{ij} y_i \tag{4.1}$$

The hidden-layer neurons apply a sigmoid activation function to their inputs

$$y_j = \frac{1}{1 + e^{-x_j}} \tag{4.2}$$

while the output neuron is linear. The weights are initially set to random values and the ANN's output o_p calculated for each event p. The weights are then adjusted to minimise the error function

$$E = \sum_p e_p \tag{4.3}$$

$$e_p = \frac{1}{2}(o_p - t_p)^2 \tag{4.4}$$

where t_p is the target output (in our case the correct identification of the event as signal ($t_p = 1$) or background ($t_p = 0$)).

The learning method is the Robbins-Monro stochastic approximation [14] applied to multi-layer perceptrons:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) \tag{4.5}$$

with

$$\Delta w_{ij}(t) = -\eta(\partial e_p/\partial w_{ij} + \delta) + \varepsilon \Delta w_{ij}(t-1) \tag{4.6}$$

In this study the values of the parameters were $\eta = 0.1$ and $\delta = \varepsilon = 0$. η was kept constant rather than decaying at each epoch through the training sample [17].

The value of ANN output which gave the maximum classification accuracy for the training sample was found, and this threshold applied to the output of the network on the test sample to classify its events.

TMlpANN from the Toolkit for Multivariate Analysis (TMVA) V03-03-03[15] combined with ROOT 5.12[16] was used. TMlpANN embodies the ROOT `TMultilayerPerceptron` class with default parameters; this is in turn based upon the package MLPfit[17]. Typical training time under Scientific Linux CERN 3.0.6 was around 3 epochs per second (2.8 GHz Pentium D).

4.2 Analysis and Results

The ANN was run on all data samples used in the GEP analysis. The results for the sample with the signal-to-background equal to 1:4 are summarised in Fig. 4. It shows the classification error (i.e. the proportion of misclassified events) after training the ANN for different numbers of epochs, and the application of the resulting networks to the test samples.

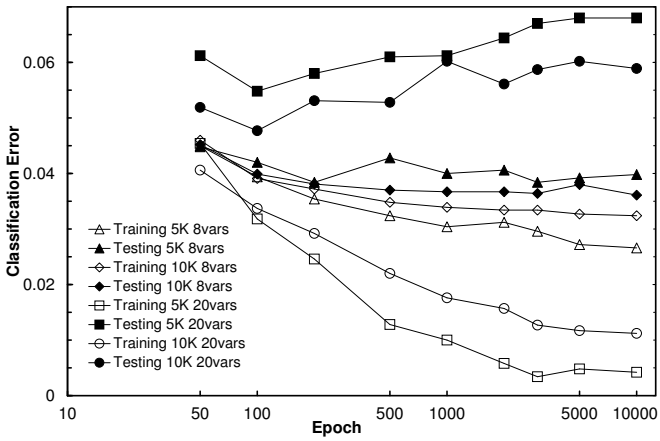

Figure 4: ANN classification performance on 5,000- and 10,000-event data samples with 8 or 20 event variables and S/B=1:4, as a function of epoch.

It can be seen that beyond epoch 100 the test error tends to increase while the training error continues to decrease, especially for the data samples with the larger number of variables and smaller sample size. This is the result of overtraining when the network becomes specific to the training sample and generalises poorly to other samples.

For the same number of variables the increase of the number of events tends to reduce the difference between the training and test errors, reducing the overtraining effect.

For the same number of events, the increase of the number of variables decreases the performance of the network, increasing the difference between the training and test errors. The increase of the number of events reduces this effect in a certain extent.

PoS(ACAT)051

These observations confirm the known behaviour of ANNs of having difficulties in dealing with irrelevant input variables, an effect that can be reduced by increasing the amount of training and test data. For the problem studied here the effect is small as the problem is of reduced complexity.

Classification Accuracy (%)							
5,000-event Samples				10,000-event Samples			
8 variables		20 variables		8 variables		20 variables	
Training	Test	Training	Test	Training	Test	Training	Test
96.1 ± 0.3	95.8 ± 0.3	96.8 ± 0.3	94.5 ± 0.3	96.1 ± 0.2	96.0 ± 0.2	96.6 ± 0.2	95.2 ± 0.2

Table 4: ANN classification accuracies after 100 epochs for the dataset with S/B=1:4.

For all the data samples analysed in this study epoch 100 was chosen as the evaluation point of the ANN performance. Table 4 shows the classification accuracy obtained with ANN for the data set with signal-to-background ratio equal to 1:4, results that are to be compared with the GEP results presented in Table 3.

5. Boosted Decision Trees method

5.1 Algorithm

Boosted Decision Trees method, known in the statistics field for quite some time, was only recently applied to high energy physics data [18], in the MiniBooNE experiment [19]. The authors of that study found the method superior to ANN, giving better event separation and being more stable and robust that ANN.

The method consists in applying a boosting algorithm to a set of Decision Trees [20] . The boosting algorithm is a procedure of combining weak classifiers, Decision Trees in this case, in order to obtain a more powerful classifier.

The BDT method used in this study follow the procedure from [18].

Decision Trees use successive decision nodes to categorise the events from the sample as either signal or background. At each node a single event variable is used to decide if the event is signal or background. Successive decisions on each separated sample form a tree structure with leaves at the end. An event is considered as signal or background depending on the classification of its ultimate leaf node during the training process.

Training of the Decision Trees is the process which defines the cut criteria at each node. At the root node, the variable and its cut value which gives the best separation into signal and background are determined. The sample is divided according to this criterion, and the same process applied to each subsample. Nodes continue to be subdivided until each node has fewer than a given number of events or until each leaf contains only signal or only background events.

In order to determine the optimum cut for each node a weight W_i is assigned to each event, starting with values equal to 1/N, where N is the number of events. The purity of the sample in the branch is then defined as:

PoS(ACAT)051

$$P = \frac{\sum_s W_s}{\sum_s W_s + \sum_b W_b} \tag{5.1}$$

where $\sum_{s,b}$ are the sums over signal and background events, respectively.

For a given branch the Gini index is then determined with the formula:

$$\text{Gini} = \left(\sum_{i=1}^{N} W_i \right) P(1-P) \tag{5.2}$$

The cut is optimised to minimise the sum of the Gini index for the left and right parts of the tree ($\text{Gini}_{\text{left child}} + \text{Gini}_{\text{right child}}$).

Boosting is applied by increasing the weights of the events which are initially misclassified, and forming a new decision tree for the re-weighted sample. The AdaBoost boosting method [21] is used in this study. For the mth tree, the following quantities are defined:

$$err_m = \frac{\sum_{i=1}^{N} W_i E_i}{\sum_{i=1}^{N} W_i} \tag{5.3}$$

$$\alpha_m = \beta \ln \left((1 - err_m)/err_m \right) \tag{5.4}$$

where $E_i = 0$ for correctly classified events, $E_i = 1$ otherwise, and β is usually unity.

The weights are then adjusted via

$$W_i \to W_i e^{\alpha_m E_i} \tag{5.5}$$

and renormalised such that $\sum_{i=1}^{N} W_i = 1$.

This procedure is repeated more times (typically several hundred) resulting in a "forest" of decision trees. Each event is subjected to the forest of trees and a likelihood estimator for the event to be signal or background is developed from the frequency of which the event is classified as signal or background by the trees in the forest. This estimator is then used as the criterion for dividing events into signal or background and to determine the efficiency of the classification.

Because the decision trees can become quite large and complex, they are usually "pruned", i.e. pairs of leaves collapsed back into one if this reduces some measure of complexity. In this study the complexity measure called Cost Complexity [20], given by the following expression, was used:

$$\alpha n + \sum_i N_i Q_i \tag{5.6}$$

where α is the "prune strength", a parameter adjusted by the user, n is the number of leaves, N_i is the number of events in the ith leaf and $Q_i = P(1-P)$ is the quality factor of that leaf.

The BDT implementation in the TMVA V03-03-03 package was used in this study. Typical training time for a 5,000-event data sample was around 3 decision trees per second using a 2.8 GHz Pentium D computer.

PoS(ACAT)051

5.2 Results

The BDT method implemented in TMVA is endowed with a large number of parameters which can be used to modify the analysis. Using the dataset with S/B = 1:1, 5,000 events, and 8 event variables, a search for parameters which gave the best classification accuracy was performed. These parameters were then used to define a smaller parameter space to be searched for the best performance on the datasets with other S/B ratios. The primary choices of the parameters were:

nTrees: 1000 – the number of decision trees in the forest,
nEventsMin: 10, 20, 30, 40 – the number of events at which the node splitting ceases,
nCuts: 20, 30 – the number of steps in optimising the cut for a node,
SeparationType: GiniIndex,
BoostType: AdaBoost,
Prune Method: CostComplexity.

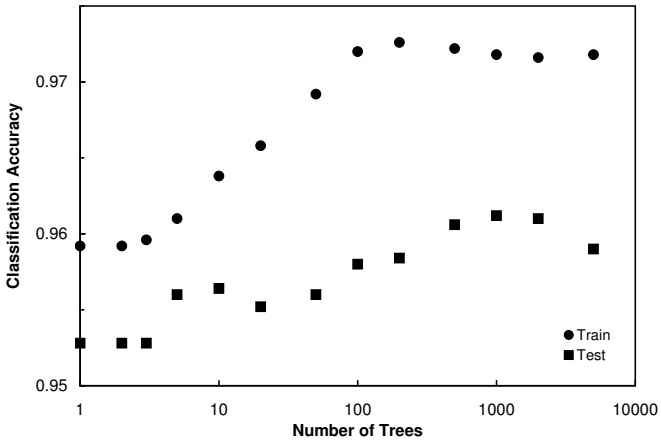

Figure 5: Evolution of the BDT classification accuracy over the increase of the size of the forest for the 5,000-event data sample, S/B=1:4, 8 variables.

An example of the evolution of the classification accuracy with the increase of the number of the decision trees in the forest is shown in Fig. 5 for the dataset with S/B = 1:4, 8 event variables and 5,000 events (for both training and test datasets). It can be seen that a plateau in the training distribution is reached around 1,000 trees. Also, the maximum value of the test classification accuracy is obtained with this configuration. This number of decision trees was found optimal for all datasets studied.

The influence on BDT classification accuracy of increasing the number of event variables from 8 to 20 and of the number of events from 5,000 to 10,000 was also studied. The results are presented in Table 5 and are to be compared with GEP and ANN results presented in Table 3 and Table 4, respectively. No significant variation of the BDT performance is seen when the number of variables or the number of events is increased. The similarity of these results indicate that BDT, as GEP, is able to deal with irrelevant input variables and, for the problem studied here, it can produce high accuracy classifications with relatively low amounts of data.

PoS(ACAT)051

Classification Accuracy (%)							
5,000-event Samples				10,000-event Samples			
8 variables		20 variables		8 variables		20 variables	
Training	Test	Training	Test	Training	Test	Training	Test
97.2 ± 0.2	96.1 ± 0.3	97.8 ± 0.2	96.1 ± 0.3	98.0 ± 0.1	96.0 ± 0.2	96.9 ± 0.2	96.1 ± 0.2

Table 5: BDT classification accuracy for the data sample with S/B=1:4

6. Comparison of the results obtained with the GEP, ANN and BDT methods

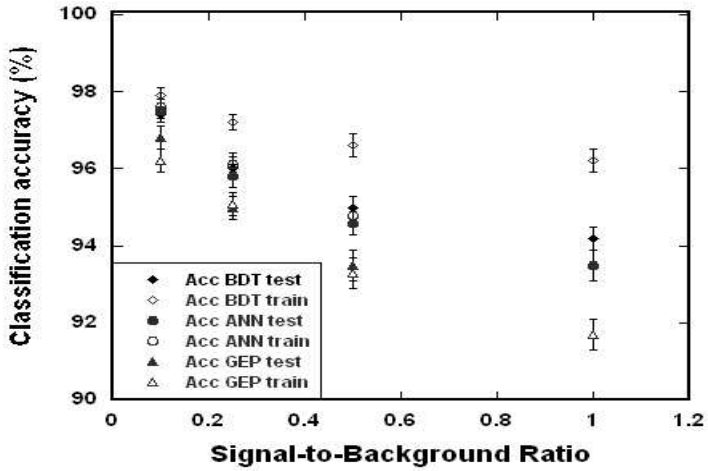

Figure 6: Comparison of the training and test classification accuracies for the GEP, ANN and BDT methods at varying S/B ratios.

The results of ANN and BDT models developed on 5,000-event samples with 8 variables at S/B ratios of 1:1, 1:2, 1:4, and 1:10 were compared to GEP results for the same samples obtained with models of head size equal to 10 and with the set of 38 input mathematical functions. The classification accuracies after training and testing for the three methods are shown in Fig. 6. The GEP and ANN training results for S/B=1 are identical to the test results (overlapping symbols on the graph). The GEP classification accuracies are with $1-3\%$ lower than those given by the other two methods, with the difference tending to increase for samples with higher S/B. The best results were obtained with the BDT methods.

It can also be noticed that the difference between the training and test results (a measure of the generalisation power of the solution) tends slightly to increase with the increase of the S/B ratio for the BDT method, to decrease for the GEP method and to remain unchanged for the ANN method.

Higher statistics studies are needed in order to extract more accurate conclusions. The limitations of the software used made such a study impractical at this moment.

PoS(ACAT)051

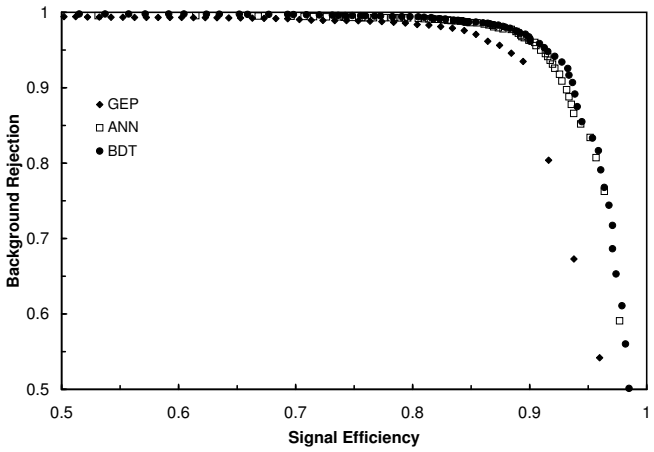

Figure 7: Background rejection as a function of signal efficiency for the three methods applied to the test 5,000-event sample with 8 variables and S/B=1:4

Different thresholds for the signal and background definition were applied on the output of the three methods (GEP with 38 input functions) for the test data sample with S/B=1:4. The corresponding signal efficiency and background rejection were calculated and their dependence is presented in Fig. 7. Signal efficiency (background rejection) was defined as the ratio of the number of events classified as signal (background) to the real number of signal (background) events in the sample. While the ANN and BDT methods are given similar results on the entire spectrum of values, the GEP methods gives lower background rejection than the other two methods at signal efficiencies over 80%. In considering this result it has to be taken into account, however, that the GEP models were developed in order to maximise the classification accuracy, signal and background events being treated equally during the development of the solution. Fitness functions that would favour the correct classification of the signal events could produce different results. These type of fitness functions were not available in the software tool used.

7. Conclusion

Gene Expression Programming was applied to an event selection problem for a set of particle physics data. For the problem and the data studied ($K_S \to \pi^+\pi^-$ decay process), the performance of the algorithm did not improve by increasing the number of event variables over 8 or by increasing the number of the input functions over 18. Also, no significant change of the classification accuracy was obtained by increasing the number of events in the data sample or by applying a parsimony pressure to the fitness function (appart of the decrease of the statistical uncertainty with the increase of the number of events).

A comparison of a Gene Expression Programming method for classifying high-energy physics data with an Artificial Neural Network and a Boosted Decision Trees method has shown that the three methods give similar test classification accuracies, with differences up to 3%. The best results seem to be obtained with Boosted Decision Trees. This conclusion, however, has to be considered valid only for the problem and the data studied here, until validations for other problems and data have been performed.

PoS(ACAT)051

It was also observed that the GEP method does not suffer from overtraining on larger data samples, or those with a large number of event variables. In addition the GEP models, as well as the BDT ones, are more easily interpreted, particularly when comparison input functions are used for GEP.

The generalisation power (the difference between the results on training and test data) of the BDT solutions shows a slight tendency to decrease toward higher S/B ratios of the input data while for the GEP solutions the tendency is the generalisation power to increase with the increase of the S/B. The ANN solutions tend to have a similar generalisation power for the S/B range investigated.

The ANN method seems slightly faster than the other methods. It requires fewer iterations to reach its optimum training point, only 100 in this study, while GEP required a few thousands generations and BDT a few hundreds of decision trees to reach their optimal solutions.

The study presented here will be extended for larger data samples and for more complex data and problems in order to consolidate the present observations concerning the advantages and disadvantages of each method. This extention is planned to use a private software implemetation of the GEP algorithm which is under development.

References

[1] J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[2] J. R. Koza, *Genetic Programming: On the Programming of the Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, 1992.

[3] C. Ferreira, *Gene Expression Programming: A New Adaptive Algorithm for Solving Problems*, Complex Systems, vol. 13, issue 2, pp. 87-129, 2001.

[4] C. Ferreira, *Gene Expression Programming: Mathematical Modelling by an Artificial Intelligence*, Angra do Heroismo, Portugal, 2002.

[5] L. Teodorescu, *Gene Expression Programming Approach to event selection in High Energy Physics*, IEEE Transaction on Nuclear Science, vol. 53, pp. 2221-2227, 2006.

[6] L. Teodorescu, *High Energy Physics Event Selection with Gene Expression Programming*, in proccedings of *Computing of High Energy and Nuclear Physics*, vol. 1, pp 215-218, 13-17 February 2006, Mumbai, India.

[7] B. Aubert et.al., *The BaBar detector*, Nuclear Instruments and Methods in Physics Research, vol. A479, pp. 1-116, 2002.

[8] T. Sjostrand, *High-energy-physics event generation with PYTHIA 5.7 and JETSET 7.4*, Computer Physics Communications, vol. 82, pp. 74-89, 1994.

[9] D. Lange, *The EvtGen particle decay simulation package*, Nuclear Instruments and Methods in Physics Research, vol. A462, pp. 152-155, 2001.

[10] S. Agostinelli et. al., *GEANT4 - a simulation toolkit*, Nuclear Instruments and Methods in Physics Research, vol. A506, pp. 250-303, 2003.

[11] D.E. Goldberg, *Genetic Algorithms in Search, Optimisation, and Machine Learning*, Addison-Wesley, 1989.

[12] http://www.gepsoft.com

PoS(ACAT)051

[13] B. Aubert et. al. (2004), *Search for Strange Pentaquark Production in e^+e^- Annihilations at $\sqrt{s} = 10.58$ GeV and in $\Upsilon(4S)$ Decays*, [arXiv.org/hep-ex/0408064].

[14] H. Robbins, and S. Monro, *A Stochastic Approximation Method*, Ann. Math. Stat., 22, 400-407, 1951.

[15] http://tmva.sourceforge.net

[16] http://root.cern.ch

[17] http://schwind.web.cern.ch/schwind/MLPfit.html

[18] B.P. Roe, H-J. Yang, J. Zhu, Y. Liu, I. Stancu, G. McGregor, *Boosted Decision Trees as an Alternative to Artificial Neural Networks for Particle Identification*, Nuclear Instruments and Methods in Physics Research, vol. A543, pp. 577-584, 2005.

[19] E. Church et. al., *BooNE Proposal*, FERMILAB-P-0898, 1997.

[20] L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone, *Classification and Regression Trees*, Wadsworth International Group, Belmont, California, 1984.

[21] Y. Freund and R.E. Schapire, *Experiments with a new boosting algorithm*, in proceedings of COLT, pp. 209-217,ACM Press, New York, 1996.

PoS(ACAT)051